Intelligent System Seminar Report

Shortest Path Algorithm by Dijkstra

Minimum Spanning Tree by Prim

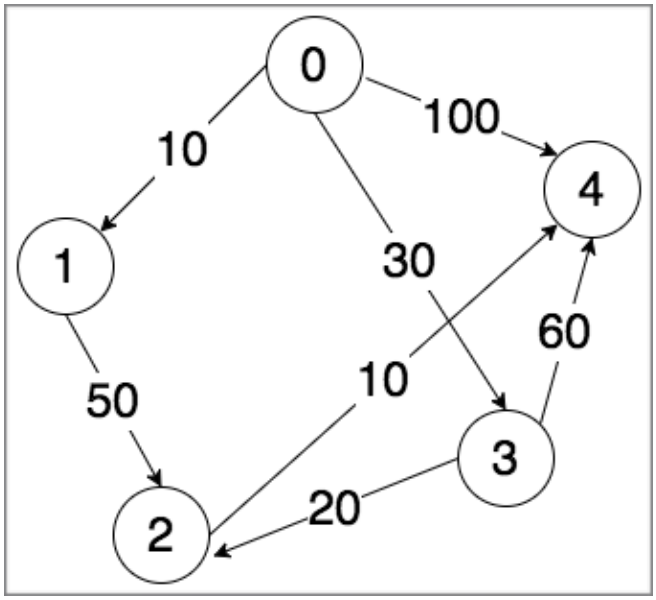
by Sinan NAR

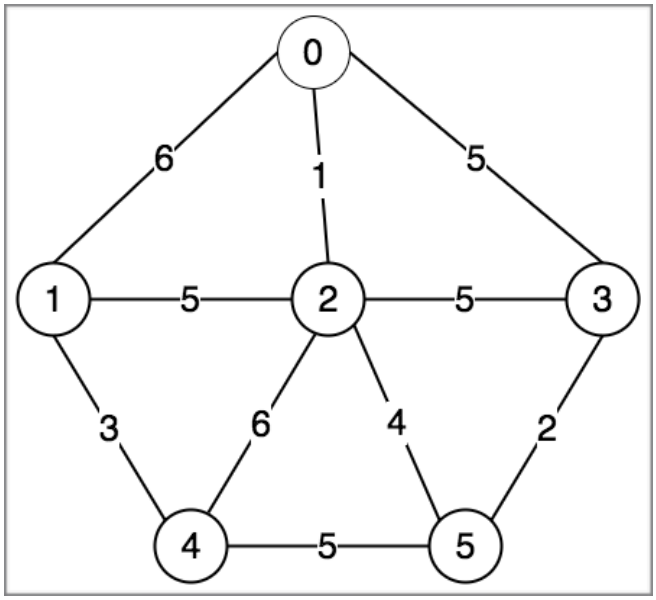
Introduction

In this seminar report, two algorithm developed by separately Dijkstra and Prim will be explained in clear and basic form.

Edsger W. Dijkstra who lived between 1930 and 2002, developed an algorithm to find shortest path from one vertex of the graph to all other vertices. Algorithm details explained in 1956 and published three years later.

Robert C. Prim who born in 1921, developed an algorithm for finding the minimum spanning tree in given graph. Actually he rediscovered and republished the algorithm developed by Czech mathematician Vojtech Jarvik in 1930. Algorithm that is going to be explained in this report, is the most basic form of Prim's algorithm and only finds minimum spanning tree in connected graphs.





Two images those can be found above, are the examples which used to explain Dijkstra and Prim's algorithms separately. Before explaining in detail, this graphs will be revisited.

Dijkstra

Breadth first search and Depth first search, two algorithm to found path on graph. Breadth first search is providing shortest path from start vertex to all other vertices assuming that the length of each edge was the same. In Dijkstra's solution, we now consider the problem of finding the shortest path where the length of each edge may be different.

For this algorithm to explain, we need two sets, S and V-S, and two array,d and p. S will contain the vertices for which have computed the shortest distance, and V-S will contain the vertices that we still need to process. The entry d[v] will contain the shortest distance from s to v, and p[v] will contain the predecessor of v in the path from s to v. This sets and arrays will be more clear after simulation.

Dijkstra's Algorithm

```
1.Initialise S with the start vertex, s, and V-S with the
                                          remaining vertices
2.for all v in V-S
3.   Set p[v] to s
4.   if there is an edge (s,v)
5.        Set d[v] to w(s,v)
6.   else
7.        set d[v] to infinite
8.while V-S is not empty
9.   for all u in V-S, find smallest d[v]
10.  Remove u fron V-S and add u to S
11.  for akk v adjacent to u in V-S
12.       if d[u] + w(u,v) is less then d[v]
13.            set d[v] to d[u] + w(u,v)
14.            set p[v] to u
```

Analysis of Dijkstra's Algorithm

Step 1 requires |V| steps.
The loop at Step 2 will be executed |v-1| times
The loop at Step 8 will be executed |v-1| times
Within the loop at step 8, we have to consider site 9 and 10. For these steps, we will have to search each value in V-S. This decreases each time through the loop at step 8.
Run time is O(|V|^2)

Related Problems and Algorithms to Dijkstra

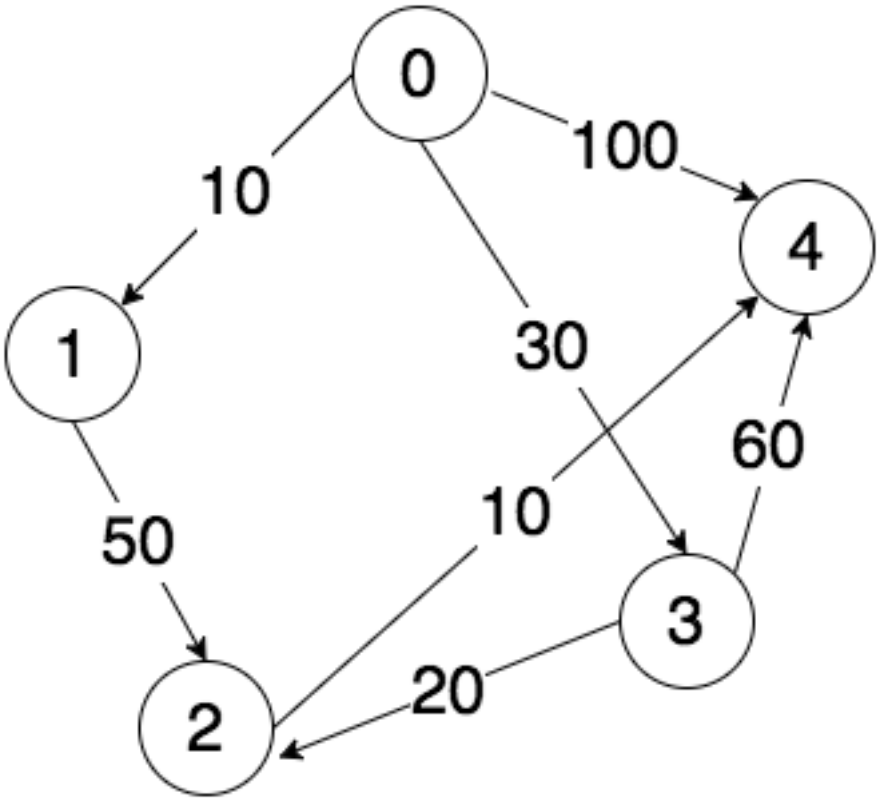
Dijkstra’s algorithm is usually the working principle behind link-state routing protocols, OSPF which stands for Open Shortest Path First and IS-IS which stands for Intermediate System to Intermediate System. Link State routing protocols are one of the two main classes of routing protocols used in packet switching networks for computer communication. IS-IS and OSPF are examples of link state routing protocols.

The A* algorithm is generalisation of Dijkstra’s algorithm that cuts down on the size of subgraph that must be explored.

The process that underlies Dijkstra’s algorithm is similar to the greedy process use in Prim's algorithm. Prim’s purpose is to find a minimum spanning tree that connects all the nodes in the graph as we will discuss later on this report. On the other side, Dijkstra is only concerned two nodes.

From a dynamic programming point of view, Dijkstra’s algorithm is a successive approximation scheme that solves the dynamic functional equation for the shortest path problem by the Reaching method.

Algorithm Simulation of Dijkstra step by step



We initialise S by placing the start vertex, s, into it. We initialise V-s by placing the remaining vertices into it. For each v in V-S, we initialise d by setting d[v] equal to the weight of w(s,v) for each vertex, v, adjacent to s and to infinite for each vertex that is not adjacent to s. We initialise p[v] to s for each v in V-S.

For the given graph, the set S would be initially {0}, V-S would be {1,2,3,4}. The arrays d and p would be defined as following table:

V	d[V]	p[V]
1	10	0
2	inf	0
3	30	0
4	100	0

For the first row shows that the distance from vertex 0 to vertex 1 is 10 and that vertex 0 is the predecessor of vertex 1. The second row shows that vertex 2 is not adjacent to vertex 0.

We no find the vertex u in V-S that has the smallest value of d[u]. Using our example, it is 1. We now consider the vertices v, that are adjacent to u. If the distance from s to u plus the distance from u to v is smaller than the known distance from s to v, then we update d[v] with smaller one and update p[v] to u. In out example, the value of d[1] is 10, and w(1,2) is 50. Since 10 + 50 = 60 is less than infinite, we set d[2] to 60 and p[2] to 1. We remove 1 from V-S and place it into S. We repeat this until V-S is empty.

After the first pass through this loop, S is {0,1} and V-S is {2,3,4} and d and p are as follows:

V	d[V]	p[V]
1	10	0
2	60	1
3	30	0
4	100	0

We again select u from V-s with the smallest d[u]. This is now 3. The adjacent vertices to 3 are 2 and 4. The distance from 0 to 3 is 30. The distance from 3 to 2 is 20. Because 30 + 20 = 50 is less than the current value of d[2], 60, we update d[2] to 50 and set p[2] to 3. Also, because pf 30 + 60 = 90 is smaller than 100, we update d[4] to 90 and set p[4] to 3 as well.

Now S is {0,1,3} and S-V is {2,4}. The arrays d and p are as follows:

V	d[V]	p[V]
1	10	0
2	50	3
3	30	0
4	90	3

Next we select vertex 2 from V-S. The only vertex adjacent to 2 is 4. Since 50 + 10 = 60 is less then 90, we update d[4] to 60 and p[4] to 2. Now S is {0,1,2,3} and V-S is {4}. d and p are as follows:

V	d[V]	p[V]
1	10	0
2	50	3
3	30	0
4	60	2

Finally we remove 4 from V-S and find that it has no adjacent vertices. We are now done. The array d shows the shortest distances from the start vertex to all other vertices, and the array p can be used to determine the corresponding paths.

For example the path from vertex 0 to vertex 4 has a length of 60, and it is the reverse of 4, 2, 3, 0, therefore, the shortest path is 0 -> 3 -> 2 -> 4.

Prim

A spanning tree is a subset of the edges of a graph such that there is only one edge between each vertex, and all of the vertices are connected. If we have spanning three for a graph, then we can access all the other vertices of the graph from the start note. The cost of a spanning tree is the sum of the weights of the edges. We want to find the minimum spanning tree or the spanning tree with the smallest cost.

For example, if we want to start up our own long distance phone company and need to connected the cities, finding the minimum spanning tree would allow us to build the cheapest network.

Overview of Prim's Algorithm

The vertices are divided into two sets: S, the set of vertices in the spanning tree, and V-S, the remaining vertices. As in Dijkstra's algorithm, we maintain two arrays: d[v] will contain the length of the shortest edge from a vertex in S to the vertex v that is in V-S, and p[v] will contain the source vertex for that edge. The only difference between the algorithm to find the shortest path and the algorithm to find the minimum spanning tree is the contents of d[v]. In the shortest path algorithm d[v] contains the total length of the path from the starting vertex. In the minimum spanning tree algorithm, d[v] contains only the length of the final edge.

Prim's Algorithm

```
1. Init S with start vertex,s, and V-S with the remaining vertices
2. for all v in V-S
3.   set p[v] to s
4.   if there is an edge (s,v)
5.        set d[v] to w(s,v)
6.   else
7.        set d[v] to infinite
8.while V-S is not empty
9.   for all u in V-S, find the smallest d[u]
10.  remove u from V0S and add it to S
11.  Insert the edge,(u,p[u]) into spanning tree
12.  for all v in V-S
13.       if w(v,u) < d[u]
14.            set d[v] to w(u,v)
15.            set p[v] to u
```

In the array d, d[v] contains the length of the shortest known edge from a vertex in S to the vertex c,while v is a member of V-S. In the array p, the value of p[v] is the source vertex of this shortest edge. When v is removed from V-S, we no longer update these entries in d and p.

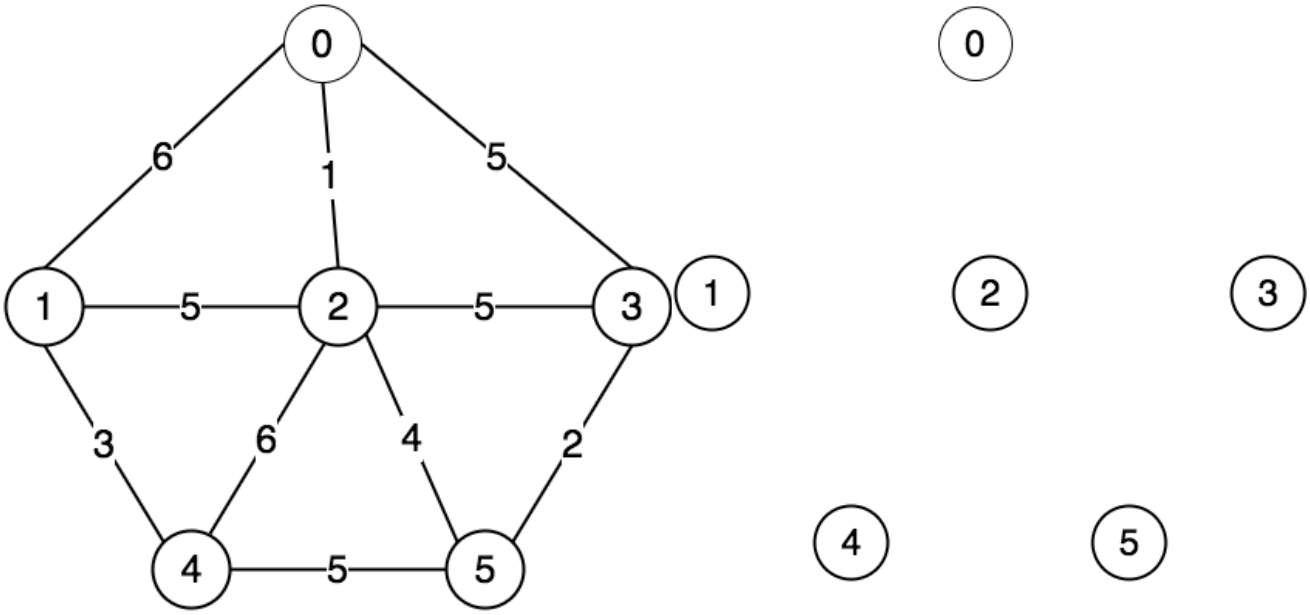
Analysis of Prim's Algorithm

Step 9 is O(|V|). Because this is within the loop at step 8, it will be executed O(|V|) times for a total time of O(|V|^2).

Overall cost of algorithm is O(|V|^2). But if we use priority queue in the algorithm, this cost will reduce to O(|E|+ |V| log (|V|)) or better performance.

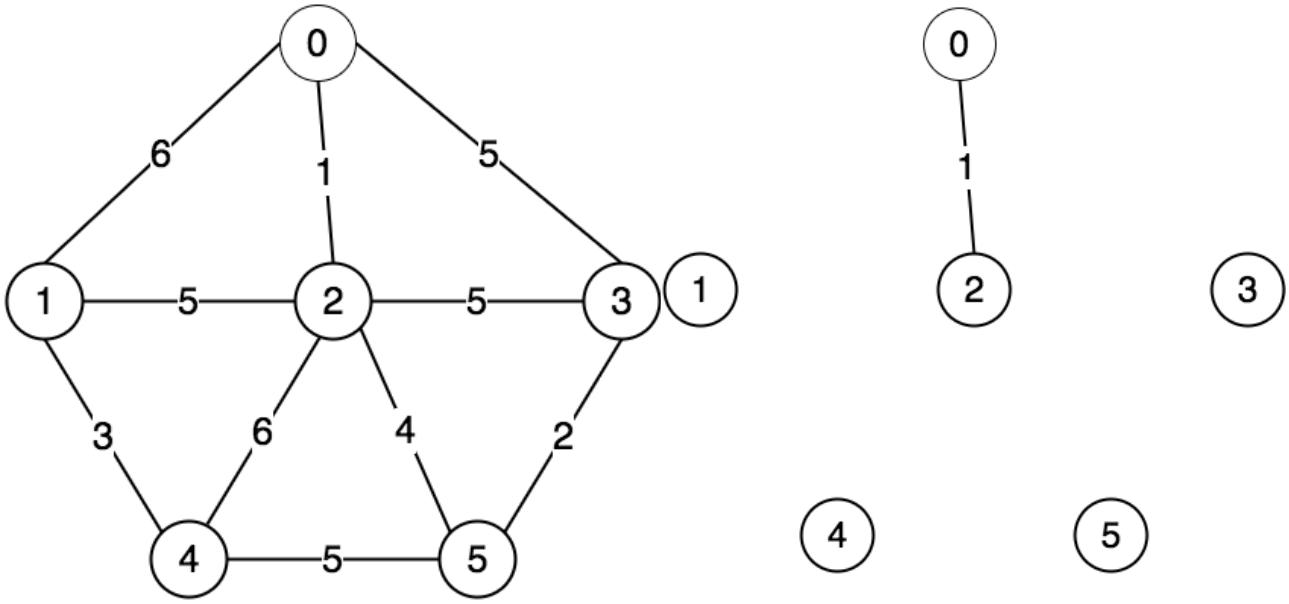
Algorithm Simulation of Prim step by step

At the beginning we do not have any path in the created graph. So at left side, we have the connected map and at the right side we have our current situation of spanning tree.

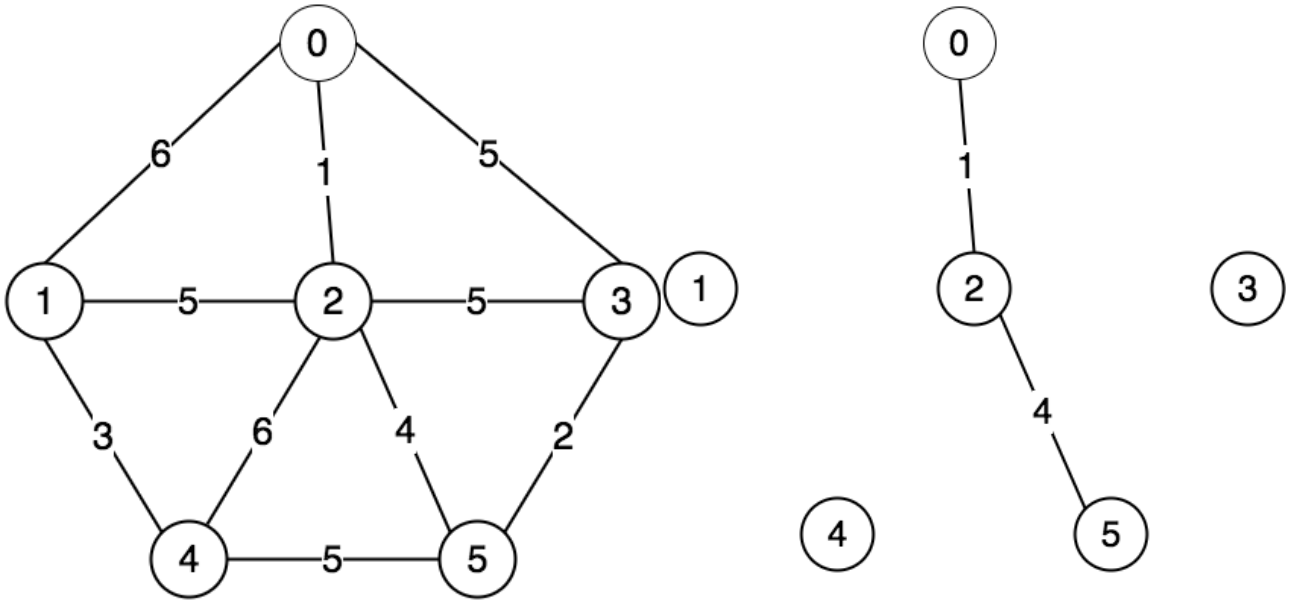


So from this given not connected spanning tree at first, 0 is our start point. We choose the smallest costed edge so it is the connection to 2. We made the connection and our spanning tree graph is going to be updated in the next figure. S = {0} and V-S = {1,2,3,4,5}.

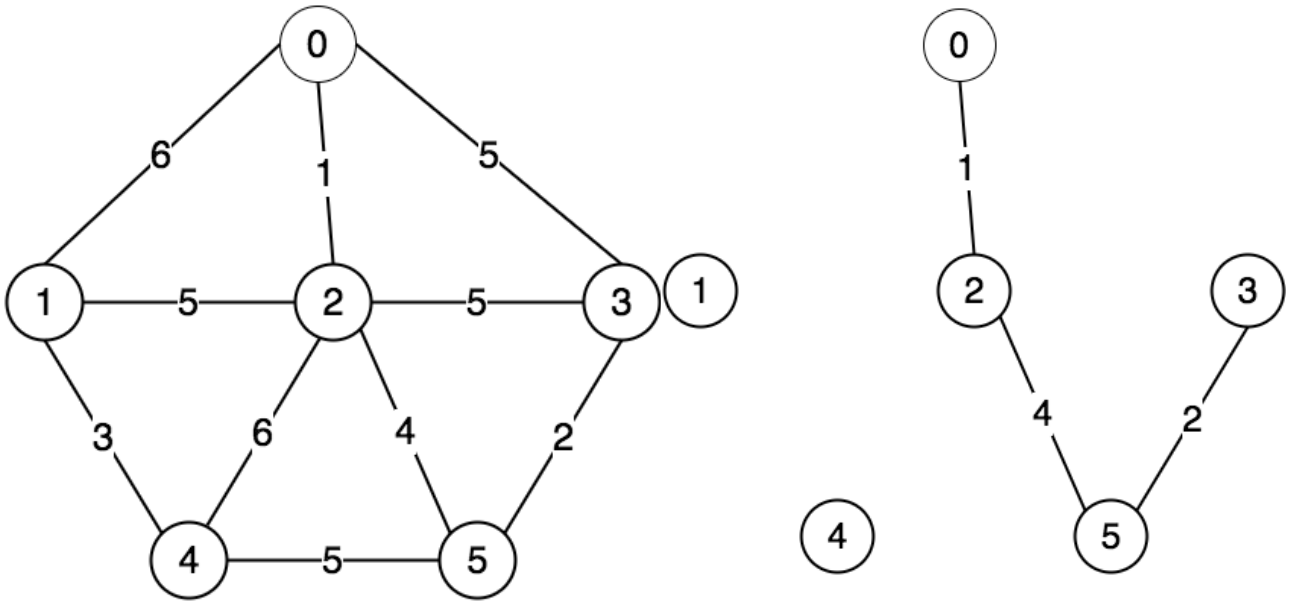
As we see at the right hand side, our spanning tree is updated. So from this point, we are going to choose smallest connection to our spanning tree.



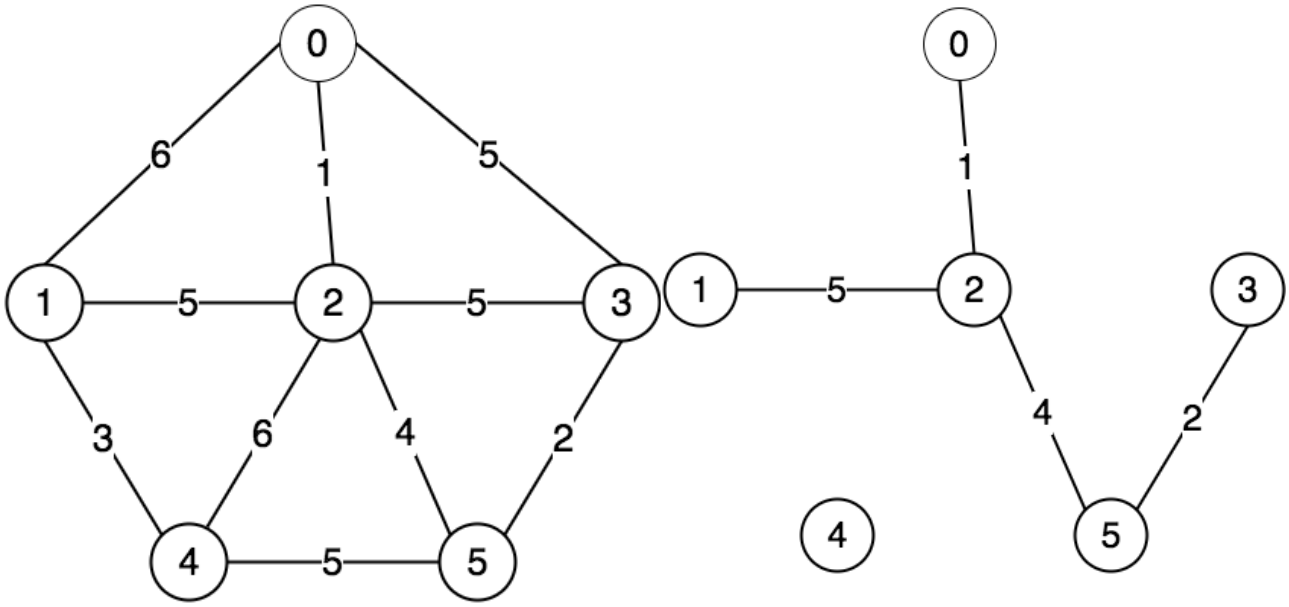
Right now S = {0,2} and V-S = {1,3,4,5}. Smallest edge between u to v, u from S and v from S-V is 2-5 connection. So our graph will be updated in this direction and our S and V-S sets are going to be updated as well.



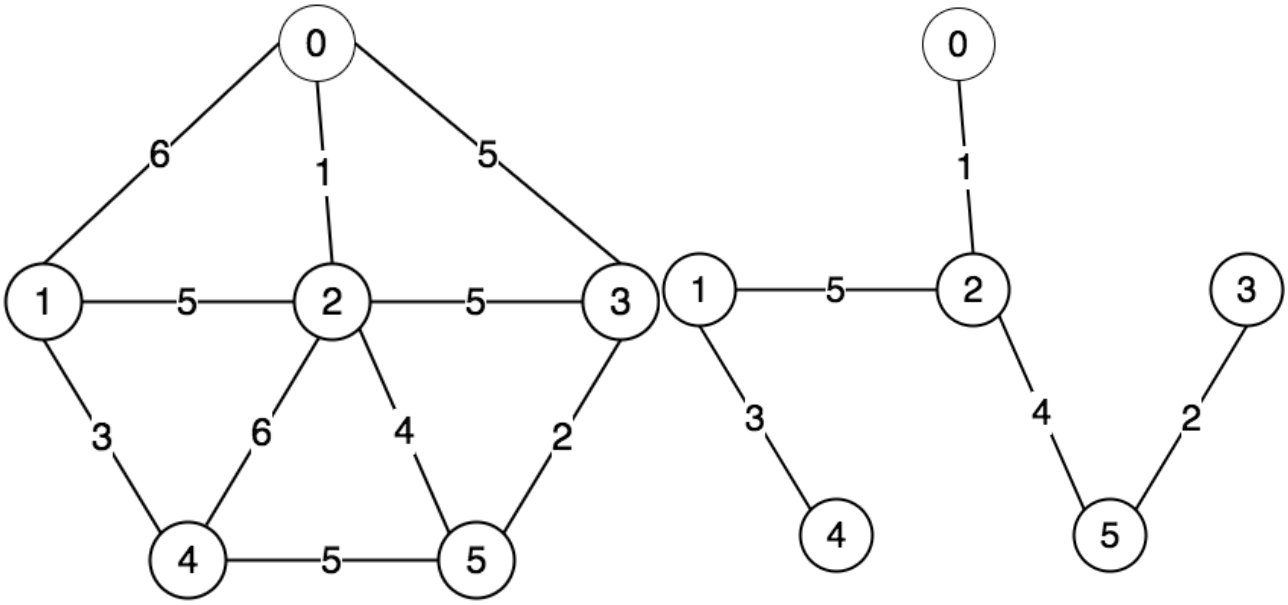
Our updated S and V-S sets are become such like S = {0,2,5} and V-S = {1,3,4} . Smallest connection on this one is 5 to 3.



Now S = {0,2,5,3} and V-S = {1,4}. There is two smallest u to v connection is appeared. First one is 1-2 and second one is 4-5. Our demonstration is going to continue with 1-2. It depends on the implementation at the end.



Now S = {0,1,2,5,3} and V-S = {4}. Smallest connection u to v is basically 1 to 4. We update out graph in the next figure.



At the end, we have emptied our V-S set and take our minimum spanning tree as graph.

References

- https://en.wikipedia.org/wiki/Dijkstras_algorithm
- https://en.wikipedia.org/wiki/Prim%27s_algorithm
- https://github.com/jimlay14/Data-Structures
- http://www.vogella.com/tutorials/JavaAlgorithmsDijkstra/article.html
- Koffman & Wolfgang; Objects, Abstraction, Data Structures and Design using JAVA

For example solution, visit github.com/sinannar